R/qtl: Just barely sustainable

Karl Broman

Biostatistics & Medical Informatics, UW–Madison

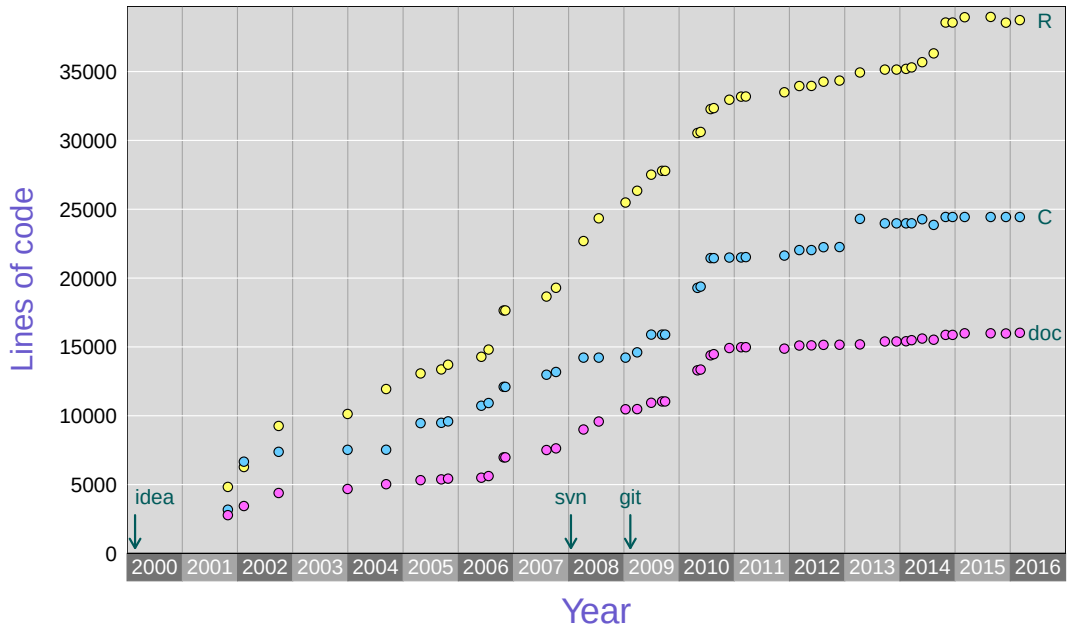
kbroman.org
github.com/kbroman
@kwbroman
Slides: bit.ly/UseR2016

These are slides for a talk that I gave at the UseR 2016 conference at Stanford on 28 June 2016.

Source: https://github.com/kbroman/Talk_UseR2016
Slides: http://bit.ly/UseR2016_nonotes
With notes: http://bit.ly/UseR2016

Also see the related paper, Broman KW (2014) Fourteen years of R/qtl: Just barely sustainable. J Open Res Softw 2(1):e11 http://doi.org/10.5334/jors.at

R/qtl



I've been developing and maintaining an R package, R/qtl, since 2000. The idea was conceived the week before R version 1.0 was released.

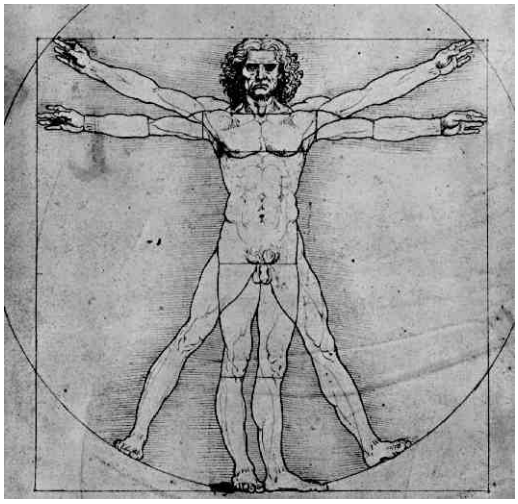
This plot shows the growth of R/qtl over time. It's currently about 60% R code and 40% C code.

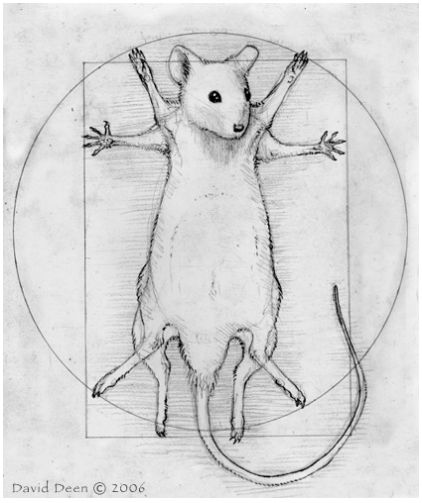
The Rd files were basically constructed by hand. Oog.

I'll start with a bit of background; hopefully I can get through this really quickly.

I focus on genetics problems, and particularly on mouse genetics.

I think these are SWR mice; the photo is from David Threadgill.



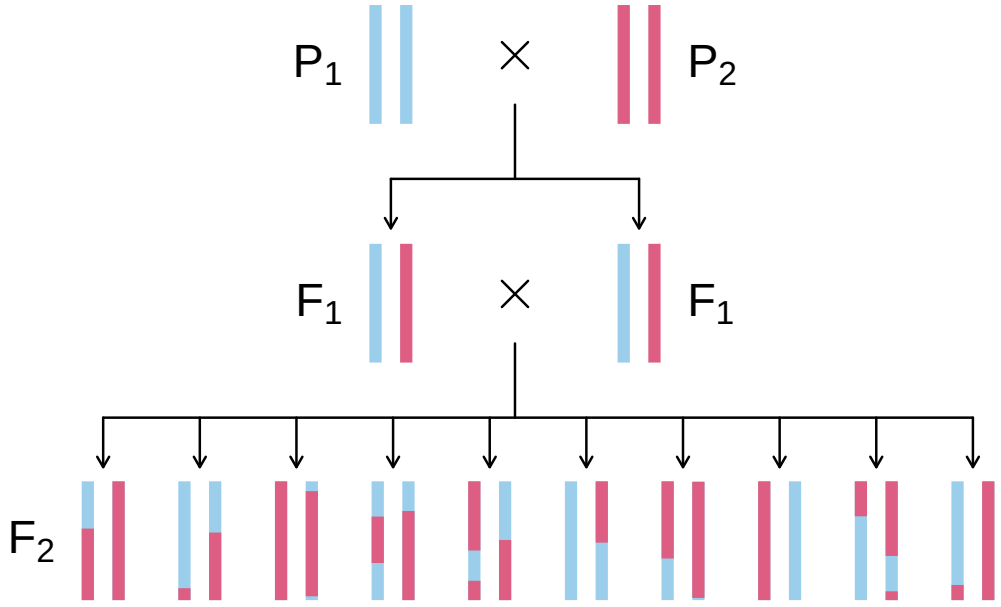


daviddeen.com

Mice are not humans, but you can learn a great deal about human biology and disease from mice.

The figure on the right is from David Deen.

Intercross



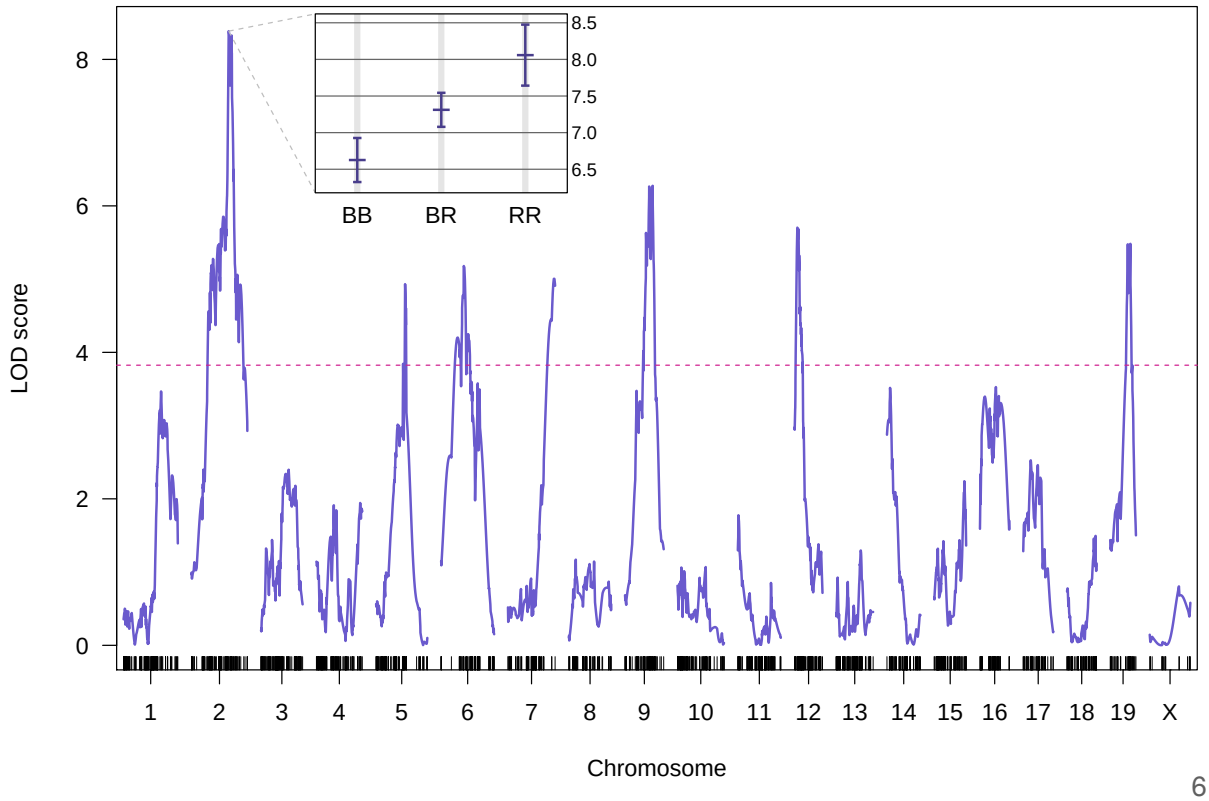
I've mostly focused on simple crosses between two inbred strains.

Say strain P_1 has low blood pressure and P_2 has high blood pressure. We cross the two strains to get the F_1 hybrid, and then intercross F_1 siblings to get a large set of F_2 individuals.

The F_2 mice may inherit a P_1 or P_2 chromosome intact, but generally their chromosomes are a mosaic of the two parental chromosomes as a result of recombination at meiosis. The points of exchange are called crossovers or recombination events.

At any one autosomal locus, the F_2 individuals will have genotype BB, BR, or RR. We'd generate many such mice and then determine their genotype along chromosomes as well as measure their phenotype (e.g., blood pressure). The simplest analyis is to look for genomic regions where genotype is associated with phenotype.

QTL mapping



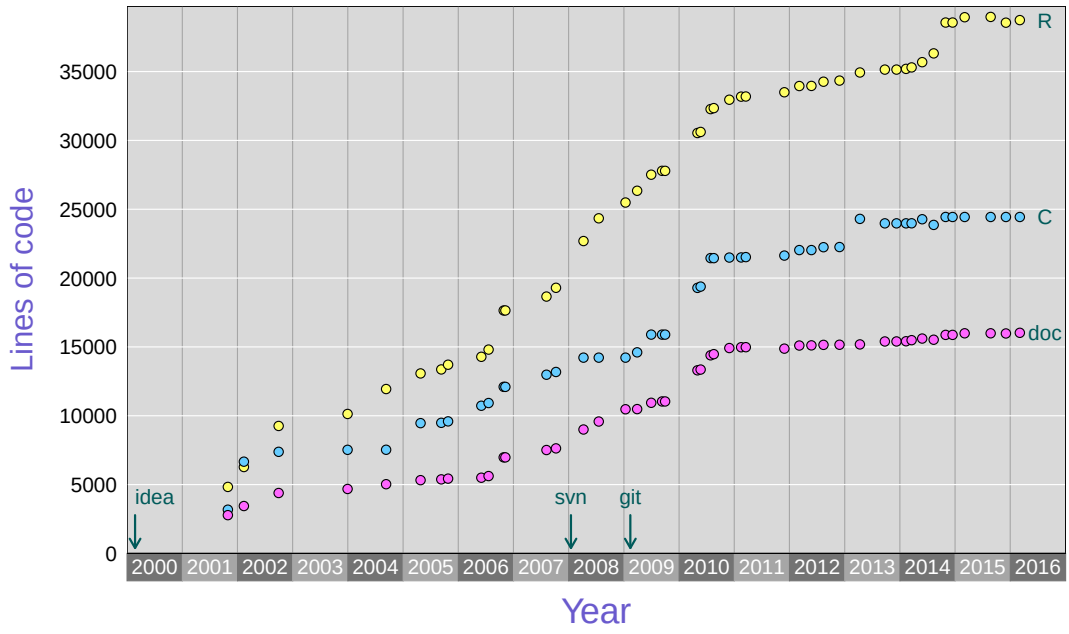
Our goal is to identify quantitative trait loci (QTL): regions of the genome for which genotype is associated with the phenotype.

The basic analysis is to consider each locus, one at a time, split the mice into the three genotype groups, and perform analysis of variance.

We then plot a test statistic that indicates the strength of the genotype-phenotype association. For historical reasons, we calculate a LOD score as the test statistic: the $\log_{10}$ likelihood ratio comparing the hypothesis that there's a QTL at that position to the null hypothesis of no QTL anywhere.

Large LOD scores indicate evidence for QTL and correspond to there being a difference in the phenotype average for the three genotype groups.

R/qtl



Back to this R/qtl package...

The goal was to have a comprehensive package for this sort of genetic analysis in model organisms.

We were particularly interested in having it be flexible and extendible: a platform for implementing new methods.

Good things

After 16 years, it becomes hard to identify things that you like about a software package.

But there are good things in R/qtl: some of the code, much of the basic user interface design, it's comprehensive, there's lots of good diagnostics and data visualizations, and it can be quite flexible.

And it seems to work.

Bad things

It's easier to identify bad things.

It's remarkable that it works, because there are basically no tests. As we'll see, there's some really bad code, and lots of repeated code. And I've had a lot of problems with memory management, particularly in copying large data sets when passed from R to C. And the code is full of if/else statements related to cross types. And the data structures can be really complicated, and they've not been well documented at all.

Stupidest code ever

```
n <- ncol(data)
temp <- rep(FALSE,n)
for(i in 1:n) {
  temp[i] <- all(data[2,1:i]=="")
  if(!temp[i]) break
}
if(!any(temp)) stop("...")
n.phe <- max((1:n)[temp])
```


This is the stupidest code ever.

It's basically trying to find the first non-blank element in a vector of character strings.

See more discussion at
`kbroman.wordpress.com/2011/08/17/the-stupidest-r-code-ever`

Input file

	A	B	C	D	E	F	G	H	I
1	liver	spleen	sex	pgm	D1Mit18	D1Mit80	D1Mit17	D2Mit379	D2Mit75
2					1	1	1	2	2
3					27.3	51.4	110.4	38.3	48.1
4	61.92	153.16	m	1	BB	SB	SB	SB	SB
5	88.33	178.58	m	1	–	–	–	BB	BB
6	58	131.91	m	1	BB	SB	SB	SB	SB
7	78.06	126.13	m	1	SB	SB	BB	SS	SS
8	65.31	181.05	m	1	–	–	–	SB	SB
9	59.26	191.54	m	1	–	–	–	SS	SS
10	59.47	154.88	m	1	BB	BB	BB	SB	SB
11	65.63	184.12	m	1	–	–	–	SB	SB
12	38.64	133.05	m	1	SB	BB	SB	SB	SB
13	60.94	275.63	m	1	–	–	–	SB	BB
14	51.48	395.25	m	1	–	–	–	SB	BB
15	47.12	260.45	m	1	BB	SB	SB	BB	BB

This is what the basic input data file looks like. That last bit of code was trying to find the first non-blank element in the second line.

Users had reported that with lots of phenotypes, it was taking hours to load their files. I thought, “Well, you’ve got lots of phenotypes.” But then the first time that I had lots of phenotypes, I realized there was a problem.

A file might take 60 seconds to load, and 58 of those seconds were spent in trying to find the first non-blank character in the second line. Urp.

Open source means
everyone can see my stupid mistakes

Version control means
everyone can see every stupid mistake I've ever made

They're both still totally worth it.

More typically bad code

The `scantwo()` function is 1446 lines long.

The related C code is 20% of the C code in R/qtl.

This sort of thing really needs to be broken up into smaller pieces.

As it is, it's really hard to extend and maintain.

Baroque data structures

```
attr(mycross$geno[["X"]]$probs, "map")
```


The data structures in R/qtl really got out of control: far too deeply nested.

Attributes can be great, but when I learned about them, I started to pile all sorts of shit in there.

User support

It's great to have one's software used, but answering their questions requires considerable patience.

Questions often have too little detail or far too much detail.

I'd hope to build a "community" of people answering each other's questions, but it continues to be largely me doing the answering.

Part of that, I think, is due to the way I use Google Groups: I filter all questions (to avoid spam), and in doing so, I tend to then just answer the questions right away. So by the time that others see the question, I've already answered it.

I try hard to be patient with, and to not be offended by, users seeking help. I'll now often let things sit for a day or two before answering.

Incorporating others' code

It's great to get contributions from others. But once you've incorporated their features, you're responsible for maintaining and supporting it.

I now think that big things should mostly be made to be separate packages.

Version control

How on earth did I get by before git?

Really critical for incorporating others' changes, and for trying out new things without breaking what's working.

Tests

How on earth does any of this work? There are basically no tests that the code works.

I'm now thoroughly in love with unit tests and Hadley's testthat.

R/qtl2: Let's not make the same mistakes

- C++ and Rcpp
- Roxygen2
- testthat
- A single "switch" for cross type
- Split into multiple packages
- Yet another data input format
- Flatter data structures, but still complex

I'm in the process of completing re-implementing R/qtl to better handle high-dimensional data and more complex crosses.

Rcpp is totally awesome, as is testthat. And OMG, how did I survive before Roxygen? All of those damned Rd files.

And I've designed things so that there's just one "switch" for cross type (an object factory in C++).

But I had the idea to split the package into multiple packages, and I'm not totally sure that was a good idea.

And I have yet another data input format, and I'm still fighting to keep my data structures from being outlandish.

It seems to me that there's a trade-off in complexity of the user interface and of the data structures.

Summary

- Sustainable software
  - Easier if you are your primary user
- Take care of your users
  - Answer each question as if it's the first
  - Primary goal is to be helpful
  - Write many focused vignettes solving real problems
- Version control
- Tests

It's always good to include a summary.

My key point here is that sustainability of software project is easy if you continue to use it as part of your own work. If the key developer moves on to other things, that's when the project will die.

Putting effort into providing help to novice users is hard but really important.

Acknowledgments

- R/qtl users
- Hao Wu, Gary Churchill, Śaunak Sen, Pjotr Prins, Danny Arends, Brian Yandell, Robert Corty, Timothée Flutre, Ritsert Jansen, Lars Ronnegard, Rohan Shah, Laura Shannon, Quoc Tran, Aaron Wolen
- NIH/NIGMS

A lot of people have contributed to R/qtl, and certainly users have been very kind to me.

Slides: bit.ly/UseR2016

kbroman.org

github.com/kbroman

@kwbroman

Here's where you can find me, as well as the slides for this talk.